Universidad de La Laguna
E.S. Ingeniería y Tecnología
Sección: Ingeniería Informática
Área: Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación
Práctica de laboratorio **#11**

1. Diseñar e implementar un Lenguaje de Dominio Específico *Domain Specific Language - DSL*, siguiendo la filosofía de Ruby que permita la definición de un examen.

   Los DSL son herramientas útiles que permiten expresar fácilmente la lógica específica de un problema particular (dominio) que de otro modo sería difícil o farragoso de escribir en otro idioma. Por lo general, se trata de definir una gramática que se asemeja más al léxico utilizado por el dominio de destino. Por ejemplo, un matemático que trabaja con matrices no piensa en bucles, iteradores o arrays, sino que piensa en términos de vectores, productos y transformaciones. El uso de un lenguaje de propósito general, como Ruby, con sólo arrays e iteradores requeriría que el matemático practicara gimnasia mental para traducir mentalmente entre el dominio de su problema y el del lenguaje con el que escribe el código (Ruby). El uso de un DSL diseñado para las operaciones que le interesan eliminaría esta traducción mental y proporcionaría un código más conciso.

   Los DSL tienen dos formas: externos e internos. Los DSL externos existen independientemente de cualquier otro lenguaje. Los DSL internos estan alojados dentro de otro lenguaje de programación - por ejemplo, Rails es un DSL interno que se aloja en el lenguaje de programación Ruby.

   El DSL que diseñe ha de permitir la definición de preguntas de una forma natural. Por ejemplo:

```
quiz = Quiz.new("Cuestionario de LPP 05/12/2014") {
  question '¿Cuántos argumentos de tipo bloque puede recibir un método?',
    right =>'1',
    wrong =>'2',
    wrong =>'muchos',
    wrong =>'los que defina el usuario'

  question "En Ruby los bloque son objetos que continen código",
    wrong =>'Cierto',
    right =>'Falso'
}
```

   Utilizar la metodología de desarrollo dirigido por pruebas (*Test Driven Development - TDD*) y la herramienta ***RSpec***.

2. Crear una cuenta en `rubygems.org`. El '*handle*' que se solicita es el nombre de usuario.

3. Puesto que ya se terminado de escribir código, se está listos para construir y publicar la gema.

   - Para *construir* la gema, desde el directorio raíz creado con Bundler ejecutar: `rake build`
   - Para crear la *version 0.1.0* de la gema ejecutar: `rake release`
   - Para *instalar* gema ejecutar: `rake install`
   - Para comprobrar que ha ido bien la instalación , ejecutar: `gem list`

4. Repartir las tareas entre los miembros del **Equipo de Trabajo**.

   Utilizar la estructura del 'directorio de trabajo del equipo' generada con *Bundler* en prácticas anteriores.

   Todos los miembros del equipo, han de realizar al menos una confirmación e incorporarla al repositorio compartido.